Software for Extracting 3D - MSSTs

Somchaipeng, Kerawit; Sporring, Jon; Kreiborg, Sven; Johansen, Peter

Publication date:
2003

Document version
Early version, also known as pre-print

Deep Structure, Singularities, and Computer Vision

DSSCV

IST-2001-35443

Deliverable

Deliverable No.:	8
Title:	Software for Extracting 3D MSSTs
Author(s):	K. Somchaipeng, J. Sporring, S. Kreiborg, and P. Johansen
Institution:	3DLab, School of Dentistry, University of Copenhagen
Classification:	Public
Date:	September 15, 2003

Abstract

The deep structure of an image is investigated, and a Multi-Scale Singularity Tree (MSST) is constructed based on the pair-wise annihilations of critical points. This report contains two main contributions. Firstly, we describe a fast, simple, and robust method of extracting feature lines from data sets of up to four dimensions, which we apply in order to extract critical paths from $3^{+1}D$ scale-spaces. Secondly, we investigate the extracting of MSSTs using either support regions or extrema partitions. Given an image, both methods produce a binary tree that mathematically represents the topological structures of the image. The software described in this report is available through the EU-project, Deep Structure, Singularities, and Computer Vision.

1 Scales, Deep Structures, and Trees

The concept of scales is applied everywhere, explicitly otherwise implicitly. Images of the same scene might look completely different when measured at different scales. Without relevant prior knowledge, we do not know what scale is better than another and all scales are equally important. We have to take all of them into account resulting in multi-scale schemes. In this report, we use a simple mathematical frame-work for working with multi scales called the Gaussian scale-space[8, 21, 10].

In practice, the finest and the coarsest scales are limited by the *inner scale*- the resolution of the measurement device and the *outer scale*- the size of the image, respectively. Between the inner and the outer scale, critical points move, annihilate and possibly are created. The movements and interactions of these critical points at all scales are called the *deep structure* of the image. It incorporates the structure of an image at all possible scales.

The deep structure depends on the dimensionality of the image[10, 3]. Only annihilations of critical points occur in one-dimensional signal, while creations of critical points are also possible for images of higher dimensionality. For generic images[1], only pair-wise creations and annihilations of critical points of opposite *Hessian signatures*[2] can occur. These annihilations and creations are called catastrophe events and the points that they occur in scale-spaces are called catastrophe points. The merging of image structures, suggests a hierarchy which can be represented mathematically using trees [11]. We call such a tree a *Multi-Scale Singularity Tree(MSST).*

The long-term goal of our work is to use the extracted MSSTs as images descriptors. The approach is relatively new in computer-vision literatures. Using trees as image descriptors transforms the computer-vision problems into tree manipulation problems, which are well-studied mathematical problems. We expect that this approach will be useful in applications, such as image matching and recognition, image database indexing, image compression, etc.

2 Introduction to Gaussian Scale-Space

The N^{+1} dimensional Gaussian scale-space $L : \mathbb{R}^{N+1} \rightarrow \mathbb{R}$ of an N dimensional image $I : \mathbb{R}^N \rightarrow \mathbb{R}$ is an ordered stack of images, where each image is a blurred version of the former. The blurring is performed according to the diffusion equation,

$$\partial_t L = \nabla^2 L, \tag{1}$$

where $\partial_t L$ is the first partial-derivative of the image in the scale direction t, and ∇^2 is the Laplace operator, which in 3 dimensions reads $\partial_x^2 + \partial_y^2 + \partial_z^2$.

[1] Images whose structure is not perturbed by noise.

[2] The sign of the determinant of the matrix of all second-order derivatives.

The Gaussian kernel is the Green's function of the heat diffusion equation, i.e.

$$L(\cdot;t) = I(\cdot) \otimes g(\cdot;t), \tag{2}$$

$$g(x;t) = \frac{1}{(4\pi t)^{N/2}} e^{-x^T x/(4t)}, \tag{3}$$

where $L(\cdot,t)$ is the image at scale t, $I(\cdot)$ is the original image, $\otimes$ is the convolution operator, $g(\cdot;t)$ is the Gaussian kernel at scale t, N is the dimensionality of the problem, and $t = \sigma^2/2$, given σ being the standard deviation of the Gaussian kernel. The *Gaussian scale-space* is the only scale-space we will consider in this report, it will henceforth be called the scale-space.

Convolution in spatial domain is equivalent to multiplication in Fourier domain,

$$I(\cdot) \otimes g(\cdot;t) = \mathcal{F}^{-1}\left(\mathcal{F}(I(\cdot))\mathcal{F}(g(\cdot;t))\right), \tag{4}$$

with $\mathcal{F}$ as the Fourier transformation operator. Further, since differentiation commutes with convolution and the Gaussian kernel is infinitely differentiable, differentiation of scale-spaces is conveniently computed,

$$\partial_{x^n} L(\cdot;t) = \partial_{x^n}\left(I(\cdot) \otimes g(\cdot;t)\right) = I(\cdot) \otimes \partial_{x^n} g(\cdot;t). \tag{5}$$

Alternative implementations of the scale-space are spatial convolutions, finite differencing schemes for the heat diffusion equation, additive operator splitting[20], and recursive implementation[4, 19]. The Fourier implementation is the most convenient for specifying derivative operators and it is precise and fast for coarse scales.

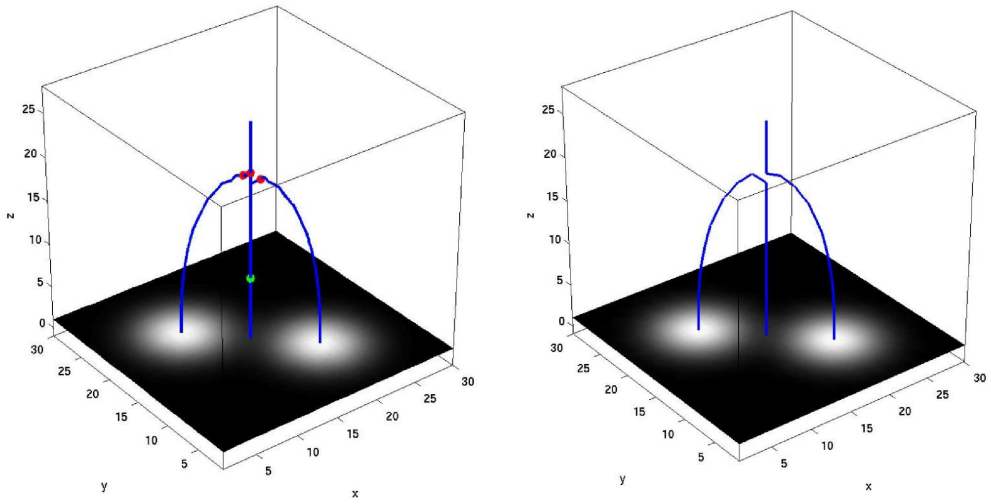
3 Computing the Deep Structure in Generic Scale-Space Images

Constructing a scale-space of an image adds a scale dimension, e.g. the scale-space of a two-dimensional image has dimensionality of three. We call such scale-space a $2^{+1}D$ scale-space. Similarly, constructing the scale-space of a three-dimensional image gives a $3^{+1}D$ scale-space. Although the dimensionality of the constructed scale-space depends on the dimensionality of the original image, critical points, namely maxima, minima and saddles, in the image at each scale are always points, and *critical paths*, which are the paths of critical points when the scale is varied, are therefore also always curves embedded in the scale-space. The following sections describe a method used to extract critical paths, the scale-space saddles and the catastrophe points on them.

At each scale in a scale-space, points with vanishing spatial gradients are called *critical points*. The type of a critical point is derived from the configuration of the eigenvalues of the Hessian matrix computed at that position. Critical points with all positive eigenvalues are minima, critical points with all negative eigenvalues are maxima, and critical points with a mixture of both negative and positive eigenvalues are saddles.

As we increase the scale parameter, the critical points move smoothly forming critical paths. At some scales, a pair of critical points with opposite Hessian signatures might meet and annihilate or be created. Such events are called catastrophe events, and the points where they occur are called catastrophe points. Generically[3], there are only two types of generic catastrophe events in scale-space namely *creation events* and *annihilation events*. It is further known that these events only occur between pairs of critical points differing in the sign of one and only one eigenvalue of the Hessian matrix.

[3]The notion of genericity is useful for regarding only the likely catastrophes, i.e. non-generic events disappear with small perturbations such as added random noise, while generic events do not.



(a) (b)

Figure 1: Critical paths in $2^{+1}D$ scale-spaces. The Marching Cubes algorithm implements choices incompatible with the scale-space structure. For this near non-generic event, when the sampling in the scale direction is not sufficiently high, the Marching Cubes algorithm produces the wrong result as shown in (a) . The error is not easy to be detected. The correct solution produced by the method described in Section 3.1 is shown in (b).

3.1 Computing Critical Paths in $2^{+1}D$ Scale-space

For $2^{+1}D$ scale-space, the critical paths are found as the intersections between surfaces of vanishing derivatives in the two spatial directions, x and y. Since the vanishing derivatives in a spatial direction are surfaces in the $2^{+1}D$ scale-space, the generic intersections are curves.

One algorithm to find the critical paths would be to calculate the surfaces of the two vanishing spatial derivatives, e.g. by using the Marching Cubes algorithm [14]. Quick to program, this is not optimal since the Marching Cubes algorithm is not tailored for the critical curves of scale-space as illustrated in Figure 1. The underlying image is a near symmetric case, where the Gaussian blobs have nearly the same intensity value at their maxima. Figure 1(a) shows the critical paths found by intersecting the surfaces produced by Marching Cubes, where it is seen that a non-generic event is suggested, where two maxima and a saddle is joined to one maximum. Figure 1(b) shows the correct structure produced by our method, where a maximum and a saddle is annihilated, and one maximum remains.

There is no need for explicit intersections between the surfaces of the two vanishing spatial derivatives. A faster, more robust, and scale-space compatible calculation of critical paths is found by linking the critical points across possibly interpolated scales. The $2^{+1}D$ image may be considered as a special kind of three-dimensional images, where the eight neighboring image points form a cube with image values on its eight vertexes. The whole scale-space is then a stack of cubes. For simplicity, we assume that there is no more than one critical path through each cube. The line segment of the critical path, which passes through a cube, is thus defined by a pair of end points on two of the cubes' six faces, as shown in Figure 2.

Our algorithm is as follows: On each face of a cube we detect the zero crossing of the spatial derivatives along the four edges of the face using inverse linear interpolation. There will be only two such points found for each type of the derivatives. Linking the detected zero crossings of the same type together gives two

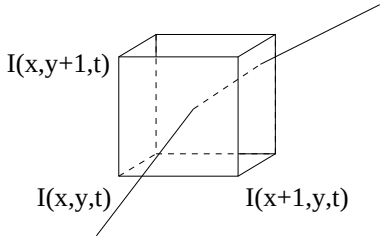


Figure 2: A critical path passing through a cube is defined as the line segment between the two points entering and exiting the cube.

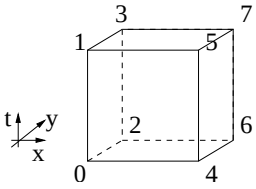


Figure 3: Our indexing of a cube for $2^{+1}D$ scale-spaces.

lines, one for the zero crossing of the first derivative in x direction and one for the zero crossing of the first derivative in y direction. The intersection point between the two lines, if exists, is one of the end points defining the critical curve passing through the cube. Since there is only one critical curve passing through the cube there will be such a point on two of the six faces of the cube. Linking them together give the critical curve passing through the cube.

The faces are indexed as shown in the cube in Figure 3. This indexing is particularly simple, since it may be represented by a three-word binary encoding, each word for the offset on each axis, x, y and t. The indexes for the four corners of the six faces are summarized in Table 1.

The extracted critical paths are now a collection of unconnected line segments. We then have to link them together forming linked critical paths. With the help of a vector of doubly linked lists, the line segments are easily joined together by comparing the positions of the two ends of a line segment with the positions of the two ends of each doubly linked list stored in the vector. If a connection is found, join the line segment with the corresponding end of the doubly linked list. If no connection can be found, add the line segment to a new doubly linked list and again to the vector as a possibly new critical path. When all line segments are stored in the vector of doubly linked list, we then have to compare the positions of the two ends of each doubly linked list with each others and join them together if possible.

In cases that there are more than one critical curve passing through a cube, we will find more than two

Free Axes	Fixed Axis	Indexes
xy	$t = 0$	0 2 4 6
xy	$t = 1$	1 3 5 7
yt	$x = 0$	0 1 2 3
yt	$x = 1$	4 5 6 7
tx	$y = 0$	0 4 1 5
tx	$y = 1$	2 6 3 7

Table 1: 3D Magic Table: Summary of the indexes for the six faces of a cube in $2^{+1}D$ scale-spaces.

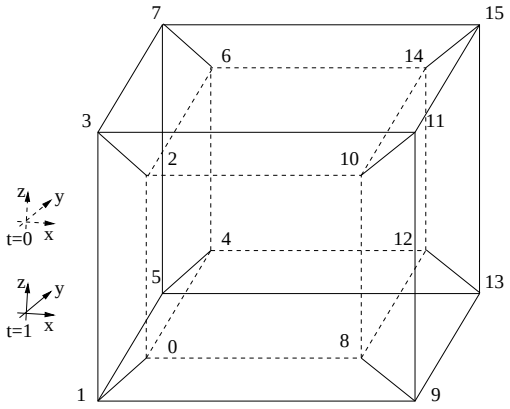


Figure 4: Our indexing of a cube for $3^{+1}D$ scale-spaces. The apparent outer cube drawn with solid lines corresponds to $t = 1$ and the inner cube drawn with dashed lines corresponds to $t = 0$.

intersection points on the six faces, possibly more than just one intersection point on each face. We then have to recursively subdivide the cube using tri-linear interpolation. The method is used to produce the critical paths in Figure 1(b). Each cube is sequentially examined, which give the computational complexity of $O(n)$, where n is the number of cubes in the $2^{+1}D$ scale-space.

3.2 Computing Critical Paths in $3^{+1}D$ Scale-space

The problem gets much more complicated in $3^{+1}D$ scale-space because human intuition often fails for dimensionality beyond three. Thanks to the reduction of dimensionality discussed in in Section 3.1, we partition a four-dimensional problem into several manageable three-dimensional subproblems.

For a $3^{+1}D$ scale-space, the sixteen neighboring points in four-dimensional space form a four-hypercube. The reader is reminded that two neighboring rectangles in two dimension share a line segment, two neighboring cubes in three dimension share a square, and two neighboring four-hypercubes share a cube. Further, there are four line segments, six squares, and eight cube around the border of a rectangle, a cube, and a four-hypercube respectively.

Similarly to $2^{+1}D$ scale-spaces, the line segment of a critical path that goes through a particular four-hypercube is defined by two points located inside two of the eight border cubes. Intersecting three surfaces of vanishing spatial derivatives in x, y, and z directions in each border cube give us such a point. For simplicity, let us assume again that there is only one critical path passing through the particular four-hypercube.

Our algorithm is thus as follows: Iteratively, two neighboring scales are calculated, together with their first spatial derivatives. For each of the eight border cubes of a four-hypercube, apply Marching Cube algorithm to extract the zero crossing surfaces of the three spatial first derivatives. Intersecting those three surfaces give a point, if exists. There will be only two such points detected inside two of the eight border cubes. Linking them together gives the line segment of the critical path passing through that four-hypercube.

In the cases that there are more than one critical path passing through the four-hypercube, there will be more than two intersection points detected in the eight border cube of that particular four-hypercube. The hyper-cube is then recursively sub-divided. The sub-dividing is not yet implemented.

As for $2^{+1}D$, we use the simple, binary numbering of the vertexes of a four-hypercube. The faces are indexed as shown in Figure 4. Table 2 summarizes the indexes of the eight vertexes for each cube at the border of a four-hypercube.

Again, the extracted line segments of the critical paths are still unconnected line segments. They have to be connected together with the help of a vector of doubly linked lists as describe in Section 3.1. Each

Free Axes	Fixed Axis	Indexes
xyz	$t = 0$	00 02 04 06 08 10 12 14
xyz	$t = 1$	01 03 05 07 09 11 13 15
yzt	$x = 0$	00 01 02 03 04 05 06 07
yzt	$x = 1$	08 09 10 11 12 13 14 15
ztx	$y = 0$	00 08 01 09 02 10 03 11
ztx	$y = 1$	04 12 05 13 06 14 07 15
txy	$z = 0$	00 04 08 12 01 05 09 13
txy	$z = 1$	02 06 10 14 03 07 11 15

Table 2: 4D Magic Table: Summary of the indexes for the eight border cubes of a four-hypercube in $3^{+1}D$ scale-spaces.

four-hypercube is sequentially examined, which give the computational complexity of $O(n)$, where n is the number of four-hypercubes in the $3^{+1}D$ scale-space. Figure 5 shows critical paths in a $3^{+1}D$ scale-space of a simple image with four light blobs.

3.3 Detecting Catastrophe Points and Scale-space Saddles on Critical Paths

Critical paths extracted using the method described in Section 3.1 and Section 3.2 are series of connected line segments, which can be traced from one end to the other end. Catastrophe points are the points where two critical points with opposite Hessian signature annihilate or be created. Therefore, the determinant of the Hessian matrix vanishes at catastrophe points. Moreover, locally, the created or annihilating critical points always move perpendicular to the scale direction at catastrophe points.

Considering a parametric representation of an extracted critical path, embedded in an $N^{+1}D$ scale-space, parameterized with $s \in [0, L]$ where L is the length of the critical path, $S(s)$ specifies the scale and $I(s)$ specified the image intensity at s along the critical path, catastrophe points are then can be detected easily as the points on critical paths where the derivative of $S(s)$ changes its sign, $\frac{d}{ds}S(s) = 0$.

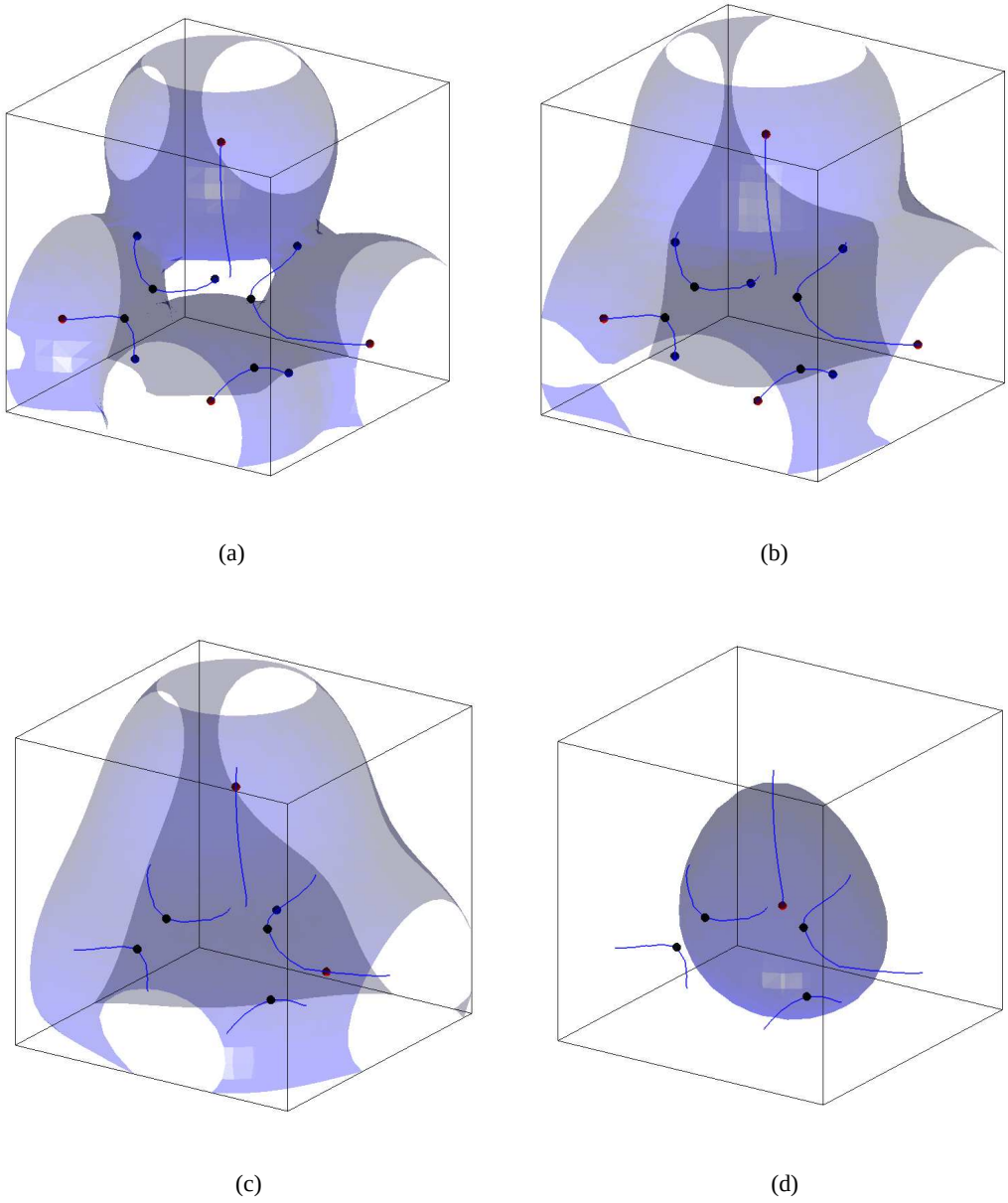
The current implementation assumes that there is no creation event, which implies that there is at maximum one catastrophe point per critical path, and it can be detected as a point on each critical path with highest scale-value. This simplification is not a major issue for our implementation.

Scale-space saddles are defined as points in scale-space with vanishing partial derivatives in both spatial and scale directions. They are indeed critical points with vanishing partial derivative in scale direction. They can be detected as points on critical paths where the derivative of the image intensity $I(s)$ changes its sign, $\frac{d}{ds}I(s) = 0$. The detection of scale-space saddles is not yet implemented.

Figure 5 shows the critical paths in a $3^{+1}D$ scale-space of a simple 3D image, where the scale axis is projected away. Four snap-shots are shown for four different scales in order to illustrate the forth dimension.

4 3D Multi-Scale Singularity Trees

A few methods of constructing tree structures from the deep structure of 2D images have been proposed in the literatures so far. In [12], the iso-photes of the intensity levels at the annihilation points are used to partition an image into regions called *extremal regions*. The nesting relations of extremal regions associated with extrema are then use to construct tree. Scale-space saddles are first introduced in [11], where the iso-surfaces kissing at these points provide a scale-space hierarchy which may be used to nest extrema in a tree.



(a) (b)

(c) (d)

Figure 5: Critical Paths in a $3^{+1}D$ scale-space. Four different scales are shown: $\sigma = 3.00$, $\sigma = 4.17$, $\sigma = 5.80$, and $\sigma = 8.07$ in (a), (b), (c), and (d) respectively. The Blue surfaces are the 2.0 iso-surfaces, the blue lines are the critical paths, the small black spheres are the catastrophe points, the small red and blue spheres show the maxima and the saddles respectively.

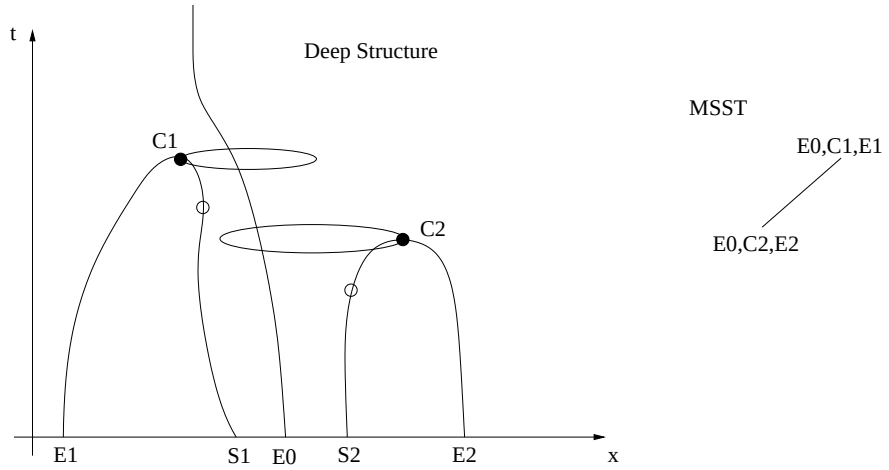


Figure 6: A simple deep structure together with its MSST. The labels 'C' denotes a catastrophe point, 'E' denotes an extremum, and 'S' denotes a saddle.

We propose two methods of constructing MSSTs. Both methods are purely coarse-to-fine methods. After the extraction of the critical paths and the catastrophe points, the methods only require the calculations of the images at the after-annihilation scales to construct the trees. Although, the methods are capable of extracting MSSTs of scale-spaces of any dimensionality, we will focus on $3^{+1}D$ scale-spaces. With the help of our new tree building scheme, both methods always produce ordered binary trees with catastrophe points as nodes, which are preferable and convenient to be subsequently processed in applications. In 3D some catastrophes are caused by creation or annihilation of saddle-saddle points. This correspond to the catastrophic interaction between a thinning and a thickening of a tube, and we have decided to ignore these catastrophes.

The first method is an extension to $3^{+1}D$ scale-spaces of the method described in [16] based on *support regions*[13] together with our new tree building scheme. The second method is a new method based on *extrema partitions*. It is inspired by the edge modeling method called *The Extrema Edges* described in [1].

4.1 MSSTs based on Support Volumes

In 2D, the border of a support region is given as the iso-phote passing through the first saddle encountered when the value of the isophote is decreased or increased from the maximum or the minimum respectively. The support regions are then never overlapped each others and each of them also contains one and only one extremum inside.

In 3D, support regions become *support volume*s. They are the volume enclosed by iso-surface passing through saddle or possibly saddles that contains only a single extremum inside. The support volumes then can be used to partition images into segments, where each segment contains exactly one extrema inside. Figure 8 shows the support volumes of the four extrema of the image in Fig 5 at scale $\sigma = 3$.

Using the catastrophe points as nodes, the parent-child relations between nodes are derived from the locations of the annihilation points and the support volumes in the images of after-annihilation scales. Moving down from the highest catastrophe in scale, when a new extremum E_c is emerged inside the support volume of an extrema E_p calculated at the after-annihilation scale, the new extrema E_c is considered as a child of the extremum E_p. The MSST is constructed as follows.

Firstly, denote the extrema together with their corresponding critical paths as E_i and the catastrophes on the paths as C_i for $i \in \{1 \dots n\}$ ordered such that the scale of a catastrophe C_i is higher than C_j when

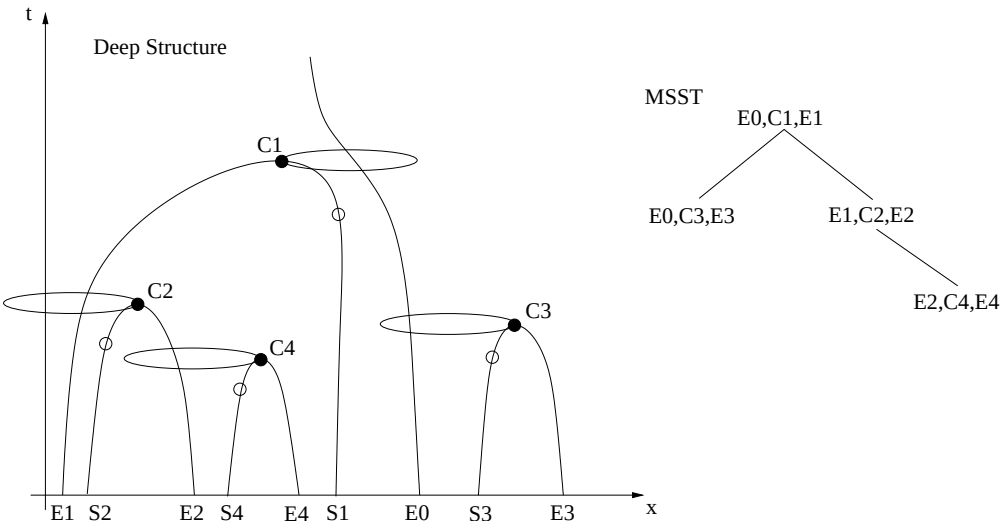


Figure 7: A complex deep structure together with its MSST. The labels 'C' denotes a catastrophe point, 'M' denotes a maximum, and 'S' denotes a saddle.

$i < j$. Discard the critical paths of saddle-saddle annihilations. Finally, by E_0 denote the critical path of an extremum that remains until infinite scale.

Each node in the tree has two ports namely the left- and the right-port, which can be used to connect the left-child and the right-child respectively. A node C_j, which has its left-port tagged as E_n can be connected as the left-child or the right-child of a node C_i if and only if the left- or the right-port of C_i is also tagged as E_n and *free*. Finally, The right-port of node C_i is always set to E_i.

The algorithm starts from the highest to the lowest catastrophe in scale. Now, initiate the MSST by:

1. Set C_1 as the root.
2. Tag the left- and the right-port as E_0 and E_1 respectively, and mark them as *free*.

Then for $i = 2 \ldots n$ repeat the following steps:

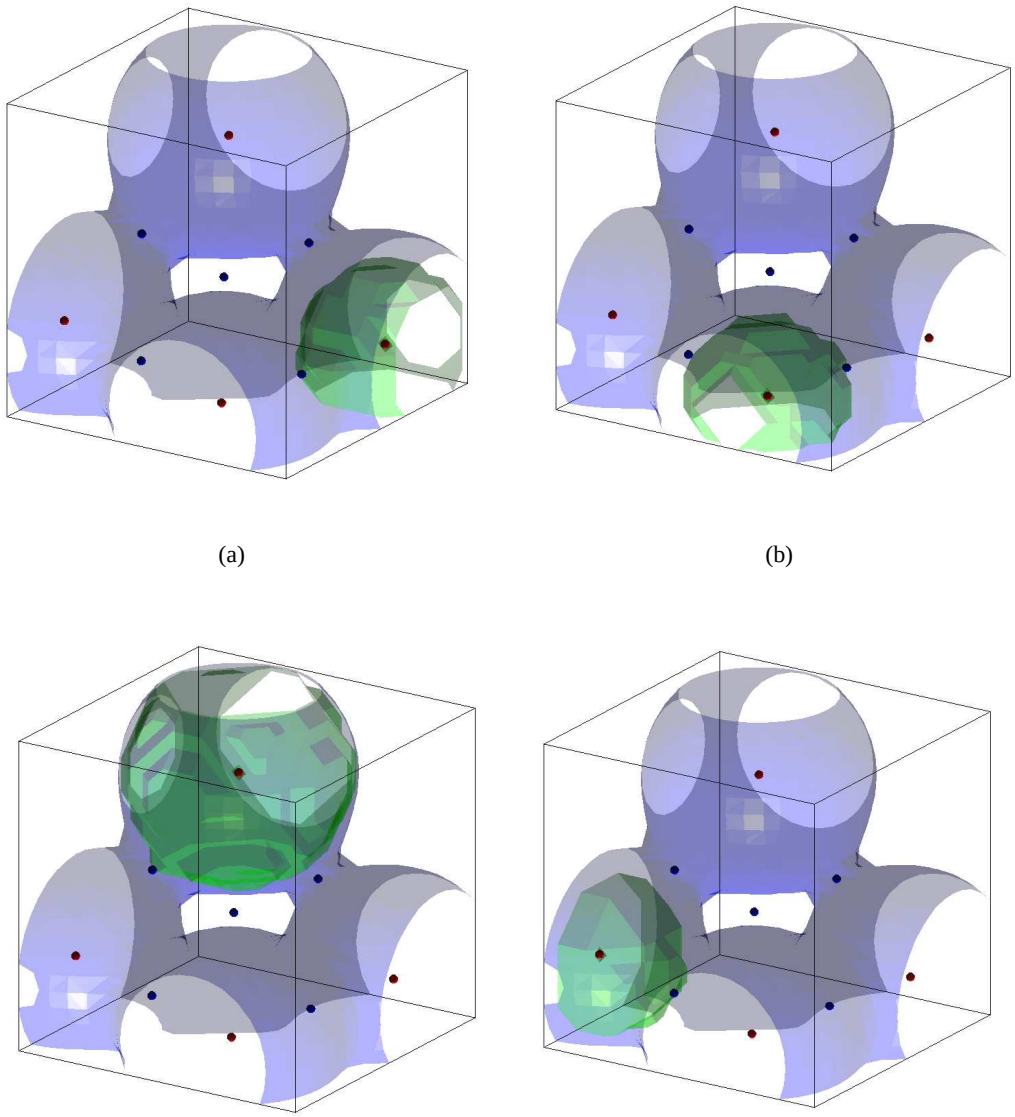
3. Calculate the support volume of all extrema at the after-annihilation scale with respect to C_i.
4. Denote E_j to be the extremum whose support volume either contains C_i or is closest to C_i. Tag the left-port of C_i as E_j, tag the right-port of C_i as E_i, and mark both as *free*.
5. Link C_i to a previous node C_k, which has a *free* left- or right-port tagged as E_j, as the corresponding left- or right-child, and mark the taken port in C_k as *occupied*.

In the end, all the critical paths in the image except for those paths that correspond to the saddle-saddle annihilation are included in the constructed tree. The resulting trees corresponding to the deep structures are shown in Figures 6 and 7.

We note that the above linking, for catastrophes not enclosed by any support volume ensures that the resulting structure will be a binary tree, and preserves the coarse to fine nature of the multi-scale singularity-tree. Neither properties are observed by linking these catastrophes to the root as suggested in [16].

Figure 9 shows an extracted MSST of the image in Fig 5. Note that one of the catastrophes is a saddle-saddle annihilation, and this is not included in the tree.

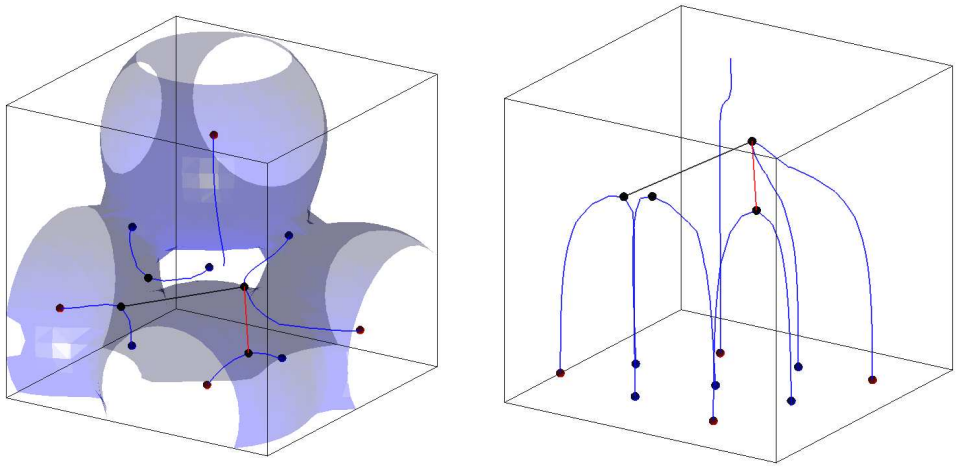
Creations are generic events in $2^{+1}D$ and $3^{+1}D$ scale-spaces. The current implementation ignores creations. However, the annihilations following creations are handled as a regular annihilation according to



(a) (b)

(c) (d)

Figure 8: Support volumes in a $3^{+1}D$ scale-space. The support volumes are calculated at scale $\sigma = 3$. Blue surfaces are the 2.0 iso-surfaces. The green surfaces show the borders of the support volumes associated with the enclosed extrema $E0$, $E1$, $E2$, and $E3$ in (a), (b), (c), and (d), respectively.



(a) (b)

Figure 9: The MSST of image in Fig. 5 based on support volumes. The blue iso-surface of the image at scale $\sigma = 3$ is shown together with the critical paths, critical points and the extracted MSST in (a), where the scale axis is projected away. The small red and blue spheres are the maxima and saddles points respectively. The blue lines are the critical path, the small black spheres are the catastrophe points. The black line and the red line denote the left-child linking and the right-child linking in the tree. The same deep structure and its MSST are shown in (b), where the z axis is projected away.

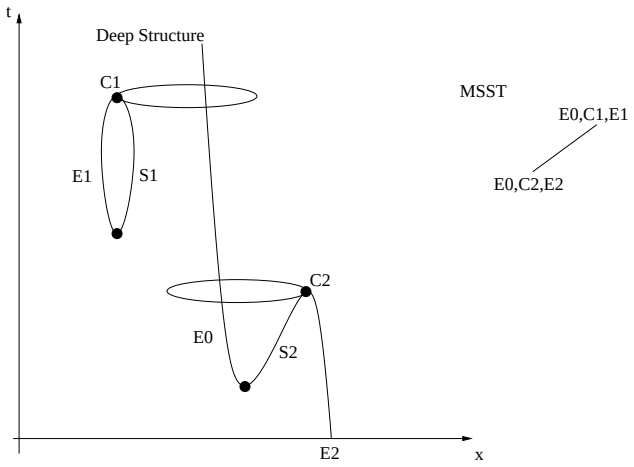


Figure 10: Creations are included by their annihilations only.

the algorithm above, see Figure 10 for details. Implication of indicated algorithm implies not all extrema can be traced to zero scale, i.e. dangling 'leaves'.

While a support volume nicely defines a region belonging to a single extremum, its separation does not necessarily reflect the local image structures. Moreover, there are parts of the image that do not belong to any support volume defined. These undefined parts of the image introduce ambiguities in the linking. The next section describes a new method of constructing MSSTs based on extrema partitions. Extrema partitions separate an image into meaningful regions associated with a single extremum and define all parts of the image totally.

4.2 MSSTs based on Extrema Partitions

Let $\Omega \subset \mathbb{R}^3$ be a compact connected domain and define $I : \Omega \to \mathbb{R}$ to be an image, $s \in \Omega$ as an extremum, and $x \in \Omega$ as a point. Consider the set of continuous functions $\gamma : [0, L] \to \Omega$ for which $\gamma(0) = s$, $\gamma(L) = x$, and $\gamma \in \Gamma_{sx}$, where Γ_{sx} is the set of all possible paths from an extremum s to a point x. The energy $E_s(x)$ with respect to an extremum s calculated at x is defined as

$$E_s(x) = \inf_{\gamma \in \Gamma_{sx}} \int_0^L |\frac{d}{dt} I(\gamma(t))| dt. \tag{6}$$

Let $S \subset \Omega$ be a set of all extrema in the image. The *extrema partition*, Z_i, associated with an extrema $i \in S$ is defined as a set of all points in the images, where the energy $E_i(x)$ is minimal,

$$Z_i = \{x \in \Omega | E_i(x) < E_j(x), \forall j \in S\}. \tag{7}$$

The energy map $M_i : \Omega \to \mathbb{R}^+$, which defines the energy at every point in the image associated with an extremum i, can be efficiently calculated using the *Fast Marching Methods* [17]. Figure 11 shows the extrema partitions of the four extrema of the image in Fig 5 calculated at scale $\sigma = 3$.

Unlike the support volumes used in the Section 4.1, which divides an image into segments using only global information by means of iso-surface, the extrema partitions divide the image into segments using both the local information derived from the derivative of the image intensity and the global information derived from the minimal path originated from the extrema. The extrema partitions and their borders include the whole image. Moreover their borders define meaningful separations, because they coincide with parts of the image, where the derivative of the intensity is high.

Using the notations of critical paths, extrema, and catastrophe points defined in the Section 4.1, the proposed algorithm starts from the highest to the lowest catastrophe in scale with the initialization of the root by:

1. Set C_1 as the root.
2. Tag the left- and the right-port as E_0 and E_1 respectively, and mark them as *free*.

Then for $i = 2 \ldots n$ repeat the following steps:

3. Calculate the energy map M_j of all extrema at the after-annihilation scale with respect to C_i.
4. Denote E_j to be the extremum whose the value of the energy map M_j evaluated at C_i is minimum among all calculated energy map. Tag the left-port of C_i as E_j, tag the right-port of C_i as E_i, and mark both as *free*.
5. Link C_i to a previous node C_k, which has a *free* left- or right-port tagged as E_j, as the corresponding left- or right-child, and mark the taken port in C_k as *occupied*.

In the end, all the critical paths in the image, except for those paths that correspond to the saddle-saddle annihilation, are included in the constructed tree. The resulting MSST of the image in Fig. 5 extracted using the method based on extrema partition is shown in Figures 12.

5 Implementation

The software is written in the C/C++ programming language[18] using the GNU g++ compiler[7] and the KDevelop developing tool [9] under the GNU/Linux operating system (Redhat 9.0). The code is documented using Doxygen[5], please see Appendix A for information about the documentation. OpenGL [22] is used to visualizing the scale-spaces, detected critical points, computed critical paths, catastrophe points, as well as the extracted MSSTs. The code is published using CVS[2] on dsscv@bellis.lab3d.odont.ku.dk under the project DSSCV, and can be obtained using Secure Shell[15] with proper access rights. Please contact the authors for information about obtaining the code and its documentation.

The software constructs scale-spaces using the Fourier implementation. Since we focus mainly on applications of medical images, we assume that all images are surrounded by a zero background. Hence prior to the transformation, images are padded with a band of zeroes that is 4σ wide. We use a highly optimized Fourier transform implementation called *Fastest Fourier Transform In the West(FFTW)*[6], which also supports machines equipped with multi processors.

The following point list contains known limitations and possible improvements to our implementation:

- A more sophisticated command line parser will provide more flexibility and allow user to supply a series of commands written in a script file. The use of Graphical user interface will be kept at minimum.
- The implementation of the scale-space saddle detection as described in Section 3.3 will provide more visual information of the extracted critical paths.
- The support volume is computed using recursive implementation which limits the size of the images to be processed.
- Several optimizations could be applied in order to reduce the computational time in the calculation of the energy maps. For example, the calculation may be stopped as soon as the values of the required points are certain.

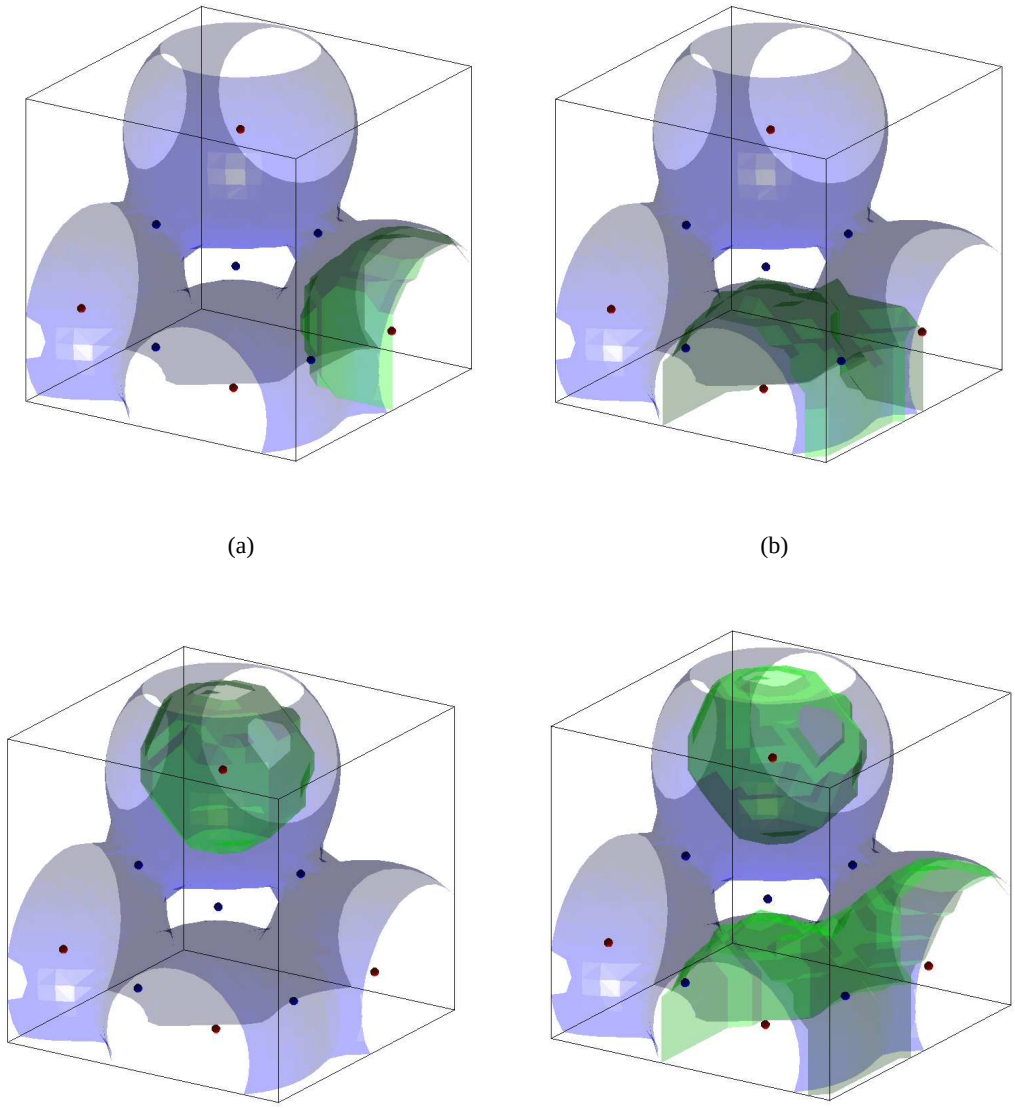
5.1 Mini-tutorial

The software is a single binary file, which can be started from a shell prompt with the command:

```
./ext3dmsst.
```

At start a new graphic window is opened, and release information is printed on the console. The user-interface of the software is a text-mode output console together with an input-output graphic window. The graphic window is used to draw lines, points, and surfaces in three-dimensional space. By default, the positive x-, y-, and z-axis are pointing to the right of the user, into the screen, and to the top of the screen respectively. Note that the program only takes input via the graphic window, therefore the graphic window has to be activated when the user wishes to supply input.

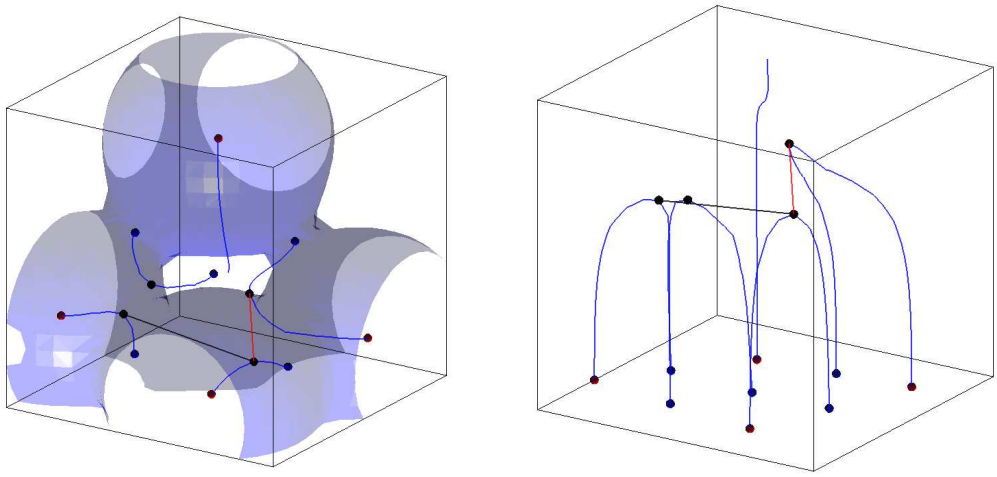
An image, supplied with the software located at `./images/data20-4.txt`, will be used in the tutorial. It is the image that is used to produce images in Figure 5. The file format is very simple. It is a text file starting with three numbers specifying the dimensions of the image followed by data separated with spaces. The images are restricted to having sizes $x \times y \times z$, were x, y, and z are even numbers.



(a) (b)

(c) (d)

Figure 11: Extrema partitions in a $3^{+1}D$ scale-space. The extrema partitions are calculated at scale $\sigma = 3$. Blue surfaces are the 2.0 iso-surfaces. The green surfaces show the borders of the extrema partitions associated with the enclosed extrema $E0$, $E1$, $E2$, and $E3$ in (a), (b), (c), and (d), respectively.



(a) (b)

Figure 12: The MSST of the image in Fig. 5 based on extrema partitions. The 2.0 iso-surface of the image at scale $\sigma = 3$ is shown in blue in (a), where the scale axis is projected away. The small red and blue spheres are the maxima and saddles points respectively. The blue lines are the critical path, the small black spheres are the catastrophe points. The black line and the red line denote the left-child linking and the right-child linking in the tree. The same critical paths and catastrophe points are shown in (b), where the z axis is projected away.

To open an image, use the command key “R”, and specify the path and the filename relatively to the current directory. For the example image type the command as follows.

```
>R
Filename [string] :./images/data20-3.txt
```

For the complete list of the commands, press “?”. Table 3 summarizes the command keys and their definitions. After having loaded an image into the memory, activate the detection of critical points with the command “C”. The graphic window can be forced to redraw with the command “Ctrl+l”. The graphic window should be showing the critical points in the image. Red, green, and blue spheres denote maxima, minima, and saddles respectively.

Use the command keys “+” and “ -” to go from the current scale to the next and previous scale. Observe the movements of the critical points, some critical points might meet and annihilate. The standard deviation σ of the Gaussian kernel for scale T is calculated as follows.

$$\sigma = \sigma_o e^T. \tag{8}$$

The σ_0, as well as the scale step dT can be changed with command “s” and “d” respectively. The default values are $\sigma_0 = 3$ and $dT = 0.05$. Jumping to any scale is made possible with the command “t” followed by the value of T.

For a better insight of the image structure and its changes with respect to scales, activate the iso-surface extraction using the command “C”, and set the value of the iso-surface with command “i”. The value of 2.0 is recommended for the example image. Number keys “4”, “6”, “8” and “2” can be used to rotate the view point. Navigate the view point around to see the image from different positions. Press “5” to move back to the default view point.

To extract the critical paths press “Ctrl+e” and give the number of scales to be calculated between $T = 0$ and $T = dT * (N - 1)$. The value of 20 is suggested for the given images. When finished, press “Ctrl-l” to redraw the image. The critical path together with the catastrophe points and the critical points at current scale should be drawn on the graphic window. Press “!” to toggle between xyz, xyt, xtz and txz coordinate systems.

Extracting MSSTs is totally automatic. To extract the MSST using the method based on support volume, press “Ctrl+s”. The software will print internal calculation details and finally draw the extracted tree on the graphic window. Extracting the MSST based on extrema partitions is done using the command “Ctrl+t”. Finally, the extracted MSST can be exported using the command “W”, followed by the relative path and the filename.

6 Conclusion

Scale-spaces of three-dimensional images are necessarily huge, e.g. 32 levels of a 256^3 image in 64 bits precision (C double) requires 4 Gigabytes of memory. Further, in order to compute the multi-scale singularity-tree, we need to examine the first and second order derivatives, which results in the total need of 40 Gigabytes of memory in this example. Needless to say, today there are only few computers that can handle such large amounts of memory within reasonable time. Luckily, we do not need to store the full scale-space simultaneously, our implementation sequentially examines two neighboring scales at a time, reducing the memory requirement by a factor of 16 in the above example. The memory can be further reduced for the exchange of processor speed, but such a compromise has not yet been needed, since we have chosen to work with smaller images.

The current implementation ignores two generic events: creations and annihilations of saddle pairs in $3^{+1}D$ scale-spaces. Creations are ignored since their inclusion would result in graphs instead of binary

Command Key	Definition
Basic Commands	
?	Help
l	List the Parameters
8,4,6,2	Change the View Point
5	Reset the View Point
Ctrl+l	Redraw the Graphic Window
Ctrl+x	Cancel Input
ESC	Exit
File Commands	
R	Read an Image
W	Export the extracted MSST
Surface Commands	
i	Set the Iso-Surface Value
S	Toggle the Iso-Surfaces
X	Toggle the $\partial_x I = 0$ Surfaces
Y	Toggle the $\partial_y I = 0$ Surfaces
Z	Toggle the $\partial_z I = 0$ Surfaces
A	Toggle the Support Volume/Extremal Partition Surfaces
Single-Scale Commands	
s	Set the σ_0
t	Set the T
d	Set the dT
C	Toggle the Critical Point Detection
c	List the Extracted Critical Points
v	Get the Value of a Point
+	Increase the T
-	Decrease the T
Multi-Scale Commands	
Ctrl+e	Extract the Critical Paths
x	List the Catastrophe Points
!	Toggle the Coordinate Systems
MSST Commands	
Ctrl+v	Calculate the Support Volume
Ctrl+p	Calculate the Extrema Partition
Ctrl+t	Extract the MSST based on Extrema Partitions
Ctrl+s	Extract the MSST based on Support Volumes

Table 3: Summary of the important command keys and their definitions.

trees, and it is expected that binary trees are especially simple as object representations. The saddle pair annihilation events likewise poses problems for our algorithm, since we only use extrema to define image segments. It is our expectation that these events do not represent important medical image structures.

On a computer with double Intel Xeon 2.4GHz with 4 Gigabytes Ram the current implementation constructs a $3^{+1}D$ scale-space with 20 scales sampled exponentially from $\sigma = 3$ to $\sigma = 7.75$ and extracts the critical paths and catastrophe points of a 20^3 image with four extremum and four catastrophe points in 25 seconds. With the extracted critical paths and catastrophe points, an MSST based on support volumes and an MSST based on extrema partitions can be constructed within four seconds and 15 seconds, respectively.

7 Acknowledgment

We would like to thank Frans Kanters and the Biomedical Image Analysis group at the Department of Biomedical Engineering, Technical University of Eindhoven for allowing Kerawit Somchaipeng to visit and share in fruitful discussions.

A Sourcecode documentation by Doxygen

The documentation is generated using the documentation system called Doxygen[5]. The program is expected to be continually edited to improve the performance and to add more functionalities. In order to keep the documentation of the implementation code up to date, we decided not to include the documentation as an appendix of this report but rather as a separated report of it own and delivered as a part of the source code package.

References

[1] P. A. Arbelaez and L. D. Cohen. The Extrema Edges". In *Scale Space 2003, LNCS 2695*, pages 180–195, 2003.

[2] cvs. http://www.cvshome.org.

[3] J. Damon. Local Morse Theory for Solutions to the Heat Equation and Gaussian Blurring. *Journal of Differential Equations*, 115(2):368–401, January 1995.

[4] R. Deriche. Recursively Implementing the Gaussian and its Derivatives. In V. Srinivasan, Ong Sim Heng, and Ang Yew Hock, editors, *Proceedings of the 2nd Singapore International Conference on Image Processing*, pages 263–267. World Scientific, Singapore, 1992.

[5] Doxygen. http://www.doxygen.org.

[6] Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998.

[7] Gcc. http://gcc.gnu.org.

[8] T. Iijima. Basic theory on normalization of a pattern (in case of typical one-dimensional pattern). *Bulletin of Electrotechnical Laboratory*, 26:368–388, 1962. (in Japanese).

[9] Kdevelop. http://www.kdevelop.org.

[10] J. J. Koenderink. The Structure of Images. *Biological Cybernetics*, 50:363–370, 1984.

[11] Arjan Kuijper. *The deep structure of Gaussian scale space images*. PhD thesis, Image Sciences Institute, Institute of Information and Computing Sciences, Faculty of Mathematics and Computer Science, Utrecht University, 2002.

[12] L. M. Lifshitz and S. M. Pizer. A multiresolution hierarchical approach to image segmentation based on intensity extrema. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 12(6):529–541, 1990.

[13] T. Lindeberg. *Scale-Space Theory in Computer Vision*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, USA, 1994.

[14] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM Computer Graphics*, pages 163–169, 1987.

[15] Openssh. http://www.openssh.com.

[16] B. Platel. Multiscale Hierarchical Segmentation. Technical Report BMT-Report no. 2002-04, Department of BioMedical Engineering, Technical Universitet of Eindhoven, 2002.

[17] J. A. Sethian. Fast Marching Methods. *SIAM Review*, 41(2):199–235, 1999.

[18] Bjarne Stroustrup. *The C++ programming language*. Addison-Wesley, 3rd edition, 2000.

[19] L.J. van Vliet, I.T. Young, and P.W. Verbeek. Recursive Gaussian Derivative Filters. In *Proceedings of the 14th International Conference on Pateern Recognition ICPR'98*, volume 1, pages 509–514., Brisbane, Australia, 1998. IEEE Computer Society Press.

[20] J. Weickert, K. J. Zuiderveld, B. M. ter Haar Romeny, and W. J. Niessen. Parallel implementations of AOS schemes: A fast way of nonlinear diffusion filtering. In *Proceedings of the 4th International Conference on Image Processing*, volume 3, pages 396–399, Santa Barbara, CA, USA, October26–29 1997. IEEE Computer Society Press.

[21] A. P. Witkin. Scale–space filtering. In *Proc. 8th Int. Joint Conf. on Artificial Intelligence (IJCAI '83)*, volume 2, pages 1019–1022, Karlsruhe, Germany, August 1983.

[22] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL: Programming Guide*. Addison-Wesley, 3rd edition, 1999.